U.S. Patent Application

Attorney Docket No.: 10016253-1

METHOD AND SYSTEM FOR MAINTAINING A MODULE TYPE DEFINITION TABLE

Inventors:

Lin Yan
839 Madison Avenue
Bridgewater, NJ 08807

Roselle Fernandez
1815 Willow Springs Way
Ft. Collins, Co. 80528

Robert Andrew Phillips
5 Yellow Jacket Court
Somerville, NJ 08876

Alberto Bellotti
44 Brookside Avenue
Hawthorne, NJ 07506

Janet Gryck
15 Hudnut Lane
Belle Mead, NJ 08502

Ingird M. Holzer
920 Cricket Lane
Woodbridge, NJ 07095

METHOD AND SYSTEM FOR MAINTAINING A MODULE TYPE DEFINITION TABLE

Field Of The Invention

[0001] The invention relates to the computer processing arts. It finds particular application to a method and system for maintaining a module type definition table. It will be appreciated that the present invention will find application in any computer processing environment.

Background Of The Invention

[0002] Certain computer operating systems and applications are designed to load modules that are internally requested by the operating system, internally requested by an application, or externally requested by a supported process. Such modules may include, for example, a procedure, a function, a script, a plug-in module, a driver, or a portion of an executable application.

[0003] In some computer operating systems and/or applications, modules that can be loaded are classified according to a module type. For example, in an operating system, the module type may be used to determine how to properly support the module. In some cases, a sub-system of the operating system, such as a dynamically loadable kernel module ("DLKM") sub-system, for example, may be used to support modules of different types.

[0004] Figure 1 illustrates a prior operating system configuration **100** for handling dynamically loadable modules. A table **110** of defined module types and associated type-specific support module identifiers are included in a statically configured operating system kernel **105**. Support module identifiers may refer to support modules, such as support module **120**, that contain, for example, pre-loading logic, post-loading logic, pre-registration logic, post-registration logic and registration logic.

[0005] When the operating system **105** receives a request to load a module **115**, the operating system uses the module type definition table **110** to determine whether the module type is supported. Upon locating the correct module type entry in the

table, the operating system utilizes the support modules (e.g., support module **120**) identified in the table **110** to properly support the requested module **115**.

[0006] The operating system **105** may provide module type-specific support for only the module types defined by the table within the statically configured kernel. This type of operating system configuration is unable to dynamically load modules of types that have not been previously defined in the statically configured portion of the operating system kernel.

[0007] Accordingly, whenever a new module type is to be defined, the statically configured kernel must be reconfigured to include a definition of the new module type. The statically configured kernel must be reloaded to accommodate the newly added module type definition. The reconfiguration of the statically configured kernel results in an increased kernel size and a reloading or rebooting of the operating system.

Summary Of The Invention

[0008] Methods and systems for maintaining module type definition tables are provided. According to one embodiment, a method for maintaining a module type definition table by a static operating system kernel is disclosed which includes dynamically creating a module type definition. The method also includes updating a module type definition table to include the module type definition at the direction of the static operating system kernel.

[0009] According to another embodiment, a system for maintaining a module type definition table is disclosed. The system includes module type detection logic that detects whether a module is of an undefined module type. The system also includes module type identification logic for assigning a module type associated with the undefined module type. The system further includes support module identification logic for identifying at least one support module associated with the module. The system still further includes support module loading logic for loading the at least one identified support module, and module type definition logic for externally storing data defining the module type.

[0010] Articles of manufacture are also disclosed which include processing instructions for maintaining a module type definition table.

Brief Description Of The Drawings

[0011] In the accompanying drawings which are incorporated in and constitute a part of the specification, embodiments of the invention are illustrated, which, together with a general description of the invention given above, and the detailed description given below, serve to example various embodiments of this method and system.

[0012] Figure 1 is a block diagram illustrating an exemplary prior operating system configuration;

[0013] Figure 2 is a block diagram illustrating an exemplary operating system configuration in accordance with one embodiment of the present invention;

[0014] Figure 3 is an exemplary methodology for maintaining module type definition table in accordance with one embodiment of the present invention;

[0015] Figure 4 is a block diagram illustrating an exemplary system for maintaining a module type definition table in accordance with one embodiment of the present invention.

[0016] Figure 5 is an exemplary methodology for analyzing a module type;

[0017] Figure 6 is a an exemplary methodology for updating a module type definition table; and

[0018] Figure 7 is a an exemplary methodology for loading a module.

Detailed Description Of Illustrated Embodiment

[0019] The following includes definitions of exemplary terms used throughout the disclosure. Both singular and plural forms of all terms fall within each meaning:

[0020] "Logic", as used herein, includes but is not limited to hardware, firmware, software and/or combinations of each to perform a function(s) or an action(s), and/or to cause a function or action from another component. For example, based on a desired application or needs, logic may include a software controlled microprocessor, discrete logic such as an application specific integrated circuit (ASIC), or other programmed logic device. Logic may also be fully embodied as software.

[0021] "Software", as used herein, includes but is not limited to one or more computer readable and/or executable instructions that cause a computer or other

electronic device to perform functions, actions, and/or behave in a desire manner. The instructions may be embodied in various forms such as routines, algorithms, modules or programs including separate applications or code from dynamically linked libraries. Software may also be implemented in various forms such as a stand-alone program, a function call, a servlet, an applet, instructions stored in a memory, part of an operating system or other type of executable instructions. It will be appreciated by one of ordinary skill in the art that the form of software is dependent on, for example, requirements of a desired application, the environment it runs on, and/or the desires of a designer/programmer or the like.

[0022] Referring now to one embodiment of the present invention, there is provided a method for maintaining a table of module type definitions external to the statically configured portion of the operating system kernel. Connected therewith, the size of the static kernel and the software development time for new module types is reduced by enabling the loading and unloading of modules previously undefined without the need to reconfigure the static kernel or reboot the computer. Further, enablement of development and delivery of new module types to end users without requiring a kernel patch and/or a reboot of the end user's computer is also possible.

[0023] Figure 2 illustrates an exemplary operating system configuration **200** according to one embodiment of the present invention. In the example depicted in Figure 2, a statically configured portion of an operating system kernel **205** employs an external module type definition table **220** when servicing a request to load a module, such as modules **210** and/or **215**.

[0024] As an example of the operation of configuration **200**, operating system kernel **205** receives a request to load module A **210**. The request may originate from within the operating system or from an application or other process supported by the operating system. Upon receiving the request to load module A **210**, operating system kernel **205** identifies the module type of module A to be "WSIO" and uses that information to determine whether the module type has been defined in module type definition table **220**. The module type can be identified in any manner, for example, accessing the module and detecting the module identifier, prompting a user to input the identifier, etc. The module type definition table **220** is external to the operating system kernel. The module type associated with the identified module may be used as

an index entry into the module type definition table, as is the case in the present example. Upon determining that a module type definition **225** exists for the module type "WSIO", operating system kernel **205** loads and supports module A **210** using the predefined support modules **226** identified in the module type definition **225**.

[0025] In another example, operating system kernel **205** receives a request to load module B **215**. Upon receiving the request to load module B **215**, operating system kernel **205** identifies the module type of module B to be "NEWMODTYPE" and uses that information to determine whether "NEWMODTYPE" has been defined in the module type definition table. As noted earlier, the module type can be identified in any manner, for example, accessing the module and detecting the module identifier, prompting a user to input the identifier, etc. In this case, there is no module type definition for the module type "NEWMODTYPE" in module type definition table **220**.

[0026] Upon determining that the module type definition table **220** does not include a definition for "NEWMODTYPE", operating system **205** executes logic to dynamically add a definition of "NEWMODTYPE" to the module type definition table **220**. In one embodiment, the logic for adding the module type definition enables a user to identify the support module(s) responsible for supporting the new module type and stores a definition of the new module type "NEWMODTYPE" in the module type definition table **220**. In some cases, a requested module may have a null and/or undefined module type identifier. In such a case, a new module type identifier is preferably automatically generated or provided by an operator or user.

[0027] Once the new module type definition is stored in the module type definition table, the operating system **205** preferably uses the newly created module type definition to load the requested module. In addition, the loading of the requested module, module B **215**, may include loading one or more support modules identified in the module type definition. Thus, the new module and any required support modules are dynamically identified, defined, and loaded.

[0028] It should be appreciated that in addition to merely updating the module type definition table to include a new module type definition, one embodiment extends to updating the module type definition table in accordance with other

activities, including for example, removing a module type definition and modifying a module type definition. Further, the described embodiments are not limited to operating with a single external module type definition table. One embodiment includes a first module type definition table within the static kernel and at least a second module type definition table external from the static kernel that may be dynamically maintained.

[0029] In Figure 3 there is exemplary methodology of one embodiment **300** for maintaining a module type definition table. As illustrated, the blocks represent functions, actions and/or events performed therein. It will be appreciated that electronic and software applications involve dynamic and flexible processes such that the illustrated blocks can be performed in other sequences different than the one shown. It will also be appreciated by one of ordinary skill in the art that elements embodied as software may be implemented using various programming approaches such as machine language, procedural, object oriented or artificial intelligence techniques. It will further be appreciated that, if desired and appropriate, some or all of the software can be embodied as part of a device's operating system.

[0030] The methodology begins when a module to be loaded is identified at block **310**. Before loading the module, a module type associated with the identified module is analyzed at block **315.** The module type can be identified in any manner, for example, accessing the module and detecting a module identifier embedded within or associated with the module, prompting a user to input the identifier, etc. At block **320,** a determination is made as to whether the module type has been previously defined. The determination is made by referencing the module type definition table that is external to the operating system kernel. If the module type definition table includes an entry for the module type associated with the identified module, then the identified module may be loaded according to the module type definition at block **340**.

[0031] If the module type associated with the identified module is null and/or if the module type definition table does not include a definition for the module type associated with the identified module, the methodology continues by identifying the module type associated with the identified module at block **325**. According to one

embodiment, if the module type associated with the identified module is null, a module type identifier can be automatically generated. As another example, if the module type associated with the identified module is null, a module type identifier can be provided by an operator. In yet another example, in the event that the identified module has an associated module type that is not included in the module type definition table, the associated module type identifier may be used as the index entry for a new module type definition. In embodiments where the modules are dynamically loadable kernel modules (DLKM), the module identifiers may be DLKM type identifiers.

[0032] In addition to the module type identifier, the operator or user is prompted to provide at least one support module identifier (**330**). Each support module identifier relates to a support module which may be used to provide module type-specific support for the identified module. Such support modules may include logic for providing, for example, pre-loading support, post-loading support, pre-registration support, post-registration support and/or module registration.

[0033] According to one embodiment, each support module identifier may be a pointer or reference to the identified support module. In an alternate embodiment, each support module identifier may be a symbol name associated with the identified support module. Each support module identifier embodied as a symbol name may be resolved dynamically at run time to enable execution of the identified support module.

[0034] Upon receiving the new module type identifier and the associated support module identifier(s), the identifiers are stored to the external module type definition table at block **335**, thereby providing a module type definition which may be used when loading such a module type in the future. The identified module may then be loaded according to the newly defined module type definition at block **340**.

[0035] In Figure 4, there is illustrated an exemplary system **400** for maintaining a module type definition table in accordance with one embodiment of the present invention. The illustrated dynamic module loading system **400** may be a portion of a larger system, such as an operating system or application. The system includes module loading control logic **405** for controlling the operation of the components of the system **400**.

[0036] The module loading control logic **405** receives a request to load a module **435** and employs module type analysis logic **410** to determine whether the module type associated with module **435** has been previously defined. The module type analysis logic **410** accesses the module **435** to determine its module type, and accesses an external module type definition table **430** to determine whether the module type has been previously defined. The module type definition table is not part of a statically created kernel and may be stored in any conventional persistent storage medium, including for example, a magnetic hard drive, a RAM drive or persistent electronic memory.

[0037] If the module type of MOD A **435** has been previously defined, the module loading control logic **405** directs module loading logic **425** to load module **435** in accordance with the associated previously defined module type definition stored in module type definition table **430**. If the module type associated with MOD A **435** is null and/or not defined in the module type definition table **430**, module loading control logic **405** directs data collection logic **415** to collect the appropriate data to define the module type associated with MOD A **435**.

[0038] Data collection logic **415** collects data defining the module type of MOD A **435**. In one embodiment, data collection logic **415** prompts an operator to provide the module type identifier and any support module identifier(s) which may be associated with the module type. In an alternate embodiment, the module type identifier is retrieved from MOD A **435**. In another embodiment, the module type identifier may be automatically generated and assigned to MOD A **435**.

[0039] The data collection logic **415** provides the module type identifier and support module identifiers(s) to definition table update logic **420**. The definition table update logic **420** uses the received identifiers to update module type definition table **430** to include a definition for the module type of MOD A **435**, thereby enabling the operating system to load any module having the same module type as MOD A **435** in the future without redefining the module type, without rebooting the computer and without requiring the operating system to be recompiled or patched.

[0040] Once a definition for the associated module type has been stored in module type definition table **430**, module loading control logic **405** utilizes module loading

logic **425** to load MOD A **435**. At the conclusion of the process, module loading control logic **405** returns a response to the request to load MOD A **435**. The response indicates a completion status of the request.

[0041] Figure 5 illustrates module type analysis logic **500** in accordance with one embodiment. As in the previous example, a request to load module MOD A **435** is received by module loading control logic at block **502**. The module loading control logic initiates the identification of the module type of MOD A **435**, wherein the module analysis logic accesses the module, MOD A **435,** at bock **504** and identifies the module type. The module analysis logic includes a detection means to identify the module type or identify that the module type is undefined. In the event that the module type is undefined, the module analysis logic defines the module type. The module analysis logic may define the module type by, for example, prompting a user to enter a module type for MOD A **435**. In an alternative embodiment, the module analysis logic may define the module type by, for example, generating and assigning a module type to MOD A **435**. Upon either identifying the module type of MOD A **435**, or assigning a module type to MOD A **435**, the module analysis logic accesses the module type definition table at block **506**. The module type analysis logic uses the module type of MOD A **435** to search the module type definition table for a definition that corresponds to the module type of MOD A **435**. If the module type of MOD A **435** is defined, MOD A **435** and its support modules may be loaded and control is passed back to the module control logic to initiate loading a module. Loading a module is described in more detail below in Figure 7. If the module type of MOD A **435** is not defined in the definition table, control is passed back to the module loading control logic to initiate defining a module type and updating the module type definition table, which is illustrated in Figure 6.

[0042] Figure 6 illustrates a methodology for updating a module type definition table. Upon determining that a module type is undefined, the module loading control logic initiates definition table update logic **600**. Control is passed to the data collection logic at block **604**. The data collection logic accesses the module, for example, MOD A **435** and collects data defining the module type of MOD A **435**. As described above, for example, the data collection logic can collect data supplied to it by an operator, retrieve data from the module type identifier, etc. Once the data

collection logic collects the data, control is passed to block **606** where definition table updated logic updates the external module type definition table. The table is updated, for example, by inputting the module type identifier and the identifiers of any support modules required by the module type identifier. The updated module type definition table enables the operating system to load any module having the same module type as MOD A **435** in the future without updating the module type definition table. Upon updating the module type definition tables, control is passed back to the module loading control logic to initiate loading a module.

[0043] Figure 7 illustrates module loading logic **700** to load a module that has been defined in the external module type definition table. The module loading control logic initiates a signal to load a module, for example MOD A **435** at block **702**. The module loading logic **700** obtains the module type, of for example, MOD A **435** and accesses the module type definition table at block **704**. The module loading logic **700** locates the module type of MOD A **435** in the module type definition table and obtains the definition for the module MOD A **435**. The module loading logic utilizes the obtained definition to load MOD A **435** and any support modules that are identified in the module type definition table that are required by MOD A **435**. As described above, upon loading the module, MOD A **435**, and any supporting modules identified in the module type definition table, control is passed back to the module loading control logic, which preferably initiates a response indicating completion of the request.

[0044] In an alternative embodiment, the methodology and/or system is embodied as computer readable code stored on a computer readable medium. The code may include one or more computer/processor executable instructions that cause the computer to act in a selected manner. The computer readable medium may be an optical storage device such as a CD-ROM or DVD-ROM, a magnetic storage device such as a hard disk or floppy disk, an electronic storage device such as a memory card, RAM, ROM, EPROM, EEPROM, or flash memory, or any other storage device capable of storing computer readable processor instructions.

[0045] While the present invention has been illustrated by the description of embodiments thereof, and while the embodiments have been described in considerable detail, it is not the intention of the applicants to restrict or in any way

limit the scope of the appended claims to such detail. Additional advantages and modifications will readily appear to those skilled in the art. Therefore, the invention, in its broader aspects, is not limited to the specific details, the representative apparatus, and illustrative examples shown and described. Accordingly, departures may be made from such details without departing from the spirit or scope of the applicant's general inventive concept.